Universidade Federal Fluminense
Centro Tecnológico
Escola de Engenharia
Curso de Engenharia de Telecomunicações
Programa de Educação Tutorial

Apostila Arduino

Autores: Erika Guimarães Pereira da Fonseca
Mathyan Motta Beppu
Orientador: Prof. Alexandre Santos de la Vega

Niterói-RJ
Dezembro / 2010

Sumário

Capítulo 1

Introdução ao Arduino

O Arduino faz parte do conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália em 2005 com o objetivo de criar um dispositivo para controlar projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

Arduino é uma plataforma de computação física (são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas), baseada em uma simples placa de Entrada/Saída microcontrolada e desenvolvida sobre uma biblioteca que simplifica a escrita da programação em C/C++. O Arduino pode ser usado para desenvolver artefatos interativos *stand-alone* ou conectados ao computador através de Adobe Flash, Processing, Max/MSP, Pure Data ou SuperCollider.

Um microcontrolador (também denominado MCU) é um computador em um chip, que contém processador, memória e periféricos de entrada/saída. É um microprocessador que pode ser programado para funções específicas, em contraste com outros microprocessadores de propósito geral (como os utilizados nos PCs). Eles são embarcados no interior de algum outro dispositivo, no nosso caso o Arduino, para que possam controlar suas funções ou ações.

É um kit de desenvolvimento capaz de interpretar variáveis no ambiente e transformá-las em sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada, e atuar no controle ou acionamento de algum outro elemento eletro-eletrônico conectado ao terminal de saída. Ou seja, é uma ferramenta de controle de entrada e saída de dados, que pode ser acionada por um sensor (por exemplo um resistor dependente da luz - LDR) e que, logo após passar por uma etapa de processamento, o microcontrolador, poderá acionar um atuador (um motor por exemplo). Como podem perceber, é como um computador, que têm como sensores de entrada como o mouse e o teclado, e de saída, impressoras e caixas de som, por exemplo, só que ele faz interface com circuitos elétricos, podendo receber ou enviar informações/tensões neles.

Para um melhor entendimento, abaixo na figura 1 é possível identificar os elementos principais do circuito através de diagrama em blocos.



Figura 1.1: Diagrama de Blocos

O Arduino é baseado em um microcontrolador (Atmega), e dessa forma é logicamente programável, ou seja, é possível a criação de programas, utilizando uma linguagem própria baseada em C/C++, que, quando implementadas fazem com que o hardware execute certas ações. Dessa forma, estamos configurando a etapa de processamento.

O grande diferencial desta ferramenta é que ela é desenvolvida e aperfeiçoada por uma comunidade que divulga os seus projetos e seus códigos de aplicação, pois a concepção dela é *open-source*, ou seja, qualquer pessoa com conhecimento de programação pode modificá-lo e ampliá-lo de acordo com a necessidade, visando sempre a melhoria dos produtos que possam ser criados aplicando o Arduino.

Ele foi projetado com a finalidade de ser de fácil entendimento, programação e aplicação, além de ser multiplataforma, ou seja, podemos configura-lo em ambientes Windows, GNU/Linux e Mac OS. Assim sendo, pode ser perfeitamente utilizado como ferramenta educacional sem se preocupar que o usuário tenha um conhecimento específico de eletrônica, mas pelo fato de ter o seu esquema e software de programação *open-source*, acabou chamando a atenção dos técnicos de eletrônica, que começaram a aperfeiçoá-la e a criar aplicações mais complexas.

Capítulo 2

Características do Duemilanove

O Arduino Duemilanove (2009 em italiano) é uma placa baseada no microcontrolador ATmega168 ou ATmega328. Tem 14 pinos de entrada ou saída digital (dos quais 6 podem ser utilizados como saídas PWM), 6 entradas analógicas, um oscilador de cristal 16 MHz, controlador USB, uma tomada de alimentação, um conector ICSP, e um botão de *reset*. Para sua utilização basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC para DC ou bateria.

2.1 Características

Microcontrolador	ATmega328 ou ATmega168
Tensão operacional	5 V
Tensão de alimentação (recomendada)	7-12 V
Tensão de alimentação (limites)	6-20 V
Pinos I/O digitais	14 (dos quais 6 podem ser Saídas PWM)
Pinos de entrada analógica	6
Corrente contínua por pino I/O	40 mA
Corrente contínua para o pino 3.3 V	50 mA
Memória flash	32 KB (2KB usados para o bootloader) / 16KB
SRAM	2 KB
EEPROM	1 KB
Frequência de clock	16 MHz

Tabela com as características básicas do arduino Duemilanove.

2.2 Alimentação

O Arduino Duemilanove pode ser alimentado pela conexão USB ou por qualquer fonte de alimentação externa. A fonte de alimentação é selecionada automaticamente.

A alimentação externa (não-USB) pode ser tanto de uma fonte ou de uma bateria. A fonte pode ser conectada com um *plug* de 2,1 mm (centro positivo) no conector de alimentação. Cabos vindos de uma bateria podem ser inseridos nos pinos GND (terra) e V_{in} (entrada de tensão) do conector de alimentação.

A placa pode operar com uma alimentação externa de 6 a 20 V. Entretanto, se a alimentação for inferior a 7 V o pino 5 V pode fornecer menos de 5 V e a placa pode ficar instável. Se a alimentação for superior a 12 V o regulador de tensão pode superaquecer e avariar a placa. A alimentação recomendada é de 7 a 12 V.

Os pinos de alimentação são:

- V_{in}: entrada de alimentação para a placa Arduino quando uma fonte externa for utilizada. Você pode fornecer alimentação por este pino ou, se usar o conector de alimentação, acessar a alimentação por este pino;
- 5 V: A fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa. Pode ser proveniente do pino V_{in} através de um regulador *on-board* ou ser fornecida pelo USB ou outra fonte de 5 V;
- 3 V3: alimentação de 3,3 V fornecida pelo circuito integrado FTDI (controlador USB). A corrente máxima é de 50 mA;
- GND (ground): pino terra.

2.3 Memória

O ATmega328 tem 32 KB de memória *flash* (onde é armazenado o software), além de 2 KB de SRAM (onde ficam as variáveis) e 1 KB of EEPROM (esta última pode ser lida e escrita através da biblioteca EEPROM e guarda os dados permanentemente, mesmo que desliguemos a placa). A memória SRAM é apagada toda vez que desligamos o circuito.

2.4 Entrada e Saída

Cada um dos 14 pinos digitais do Duemilanove pode ser usado como entrada ou saída usando as funções de `pinMode()`, `digitalWrite()`, e `digitalRead()`. Eles operam com 5 V. Cada pino pode fornecer ou receber um máximo de 40 mA e tem um resistor *pull-up* interno (desconectado por padrão) de 20-50 kΩ. Além disso, alguns pinos têm funções especializadas:

- Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL. Estes pinos são conectados aos pinos correspondentes do chip serial FTDI USB-to-TTL.
- PWM: 3, 5, 6, 9, 10, e 11. Fornecem uma saída analógica PWM de 8-bit com a função analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estes pinos suportam comunicação SPI, que embora compatível com o hardware, não está incluída na linguagem do Arduino.
- LED: 13. Há um LED já montado e conectado ao pino digital 13.

O Duemilanove tem 6 entradas analógicas, cada uma delas está ligada a um conversor analógico-digital de 10 bits, ou seja, transformam a leitura analógica em um valor dentre 1024 possibilidades (exemplo: de 0 a 1023). Por padrão, elas medem de 0 a 5 V, embora seja possível mudar o limite superior usando o pino AREF e um pouco de código de baixo nível. Adicionalmente alguns pinos têm funcionalidades especializadas:

- I2C: 4 (SDA) e 5 (SCL). Suportam comunicação I2C (TWI) usando a biblioteca Wire.
- AREF. referência de tensão para entradas analógicas. Usados com analogReference().
- Reset. Envie o valor LOW para reiniciar o microcontrolador. Tipicamente utilizados para adicionar um botão de *reset* aos *shields*(placas que podem ser plugadas ao Arduino para extender suas capacidades) que bloqueiam o que há na placa.

2.5 Comunicação

Com o Arduino Duemilanove a comunicação com um computador, com outro Arduino ou com outros microcontroladores é muito simplificada. O ATmega328 permite comunicação serial no padrão UART TTL (5 V), que está disponível nos pinos digitais 0 (RX) e 1 (TX). Um chip FTDI FT232RL na placa encaminha esta comunicação serial através da USB e os drivers FTDI (incluído no software do Arduino) fornecem uma porta virtual para o software no computador. O software Arduino inclui um monitor serial que permite que dados simples de texto sejam enviados e recebidos à placa Arduino. Os LEDs RX e TX da placa piscam quando os dados estão sendo transferidos ao computador pelo chip FTDI e há conexão USB (mas não quando há comunicação serial pelos pinos 0 e 1).

A biblioteca SoftwareSerial permite comunicação serial por quaisquer dos pinos digitais do Duemilanove.

O ATmega328 também oferece suporte aos padrões de comunicação I2C (TWI) e SPI. O software do Arduino inclui uma biblioteca Wire para simplificar o uso do barramento I2C. Para usar a comunicação SPI veja o *datasheet* (folha de informações) do ATmega328.

2.6 Programação

O ambiente de programação mais indicado é o do software Arduino, que pode ser baixado no seguinte site: http://www.arduino.cc/en/Main/Software. Mais detalhes sobre a programação no capítulo Referências da linguagem usada na programação do Arduino.

2.7 *Reset* Automático

Algumas versões anteriores do Arduino requerem um *reset* físico (pressionando o botão de *reset* na placa) antes de carregar um *sketch*(o programa a ser compilado). O Arduino Duemilanove é projetado de modo a permitir que isto seja feito através do software que esteja rodando no computador conectado. Uma das linhas de controle de hardware (DTR) do FT232RL está conectada ao *reset* do ATmega328 via um capacitor de 100 microFaraday . Quando esta linha é colocada em nível lógico baixo, o sinal cai por tempo suficiente para reiniciar o chip. O software Arduino usa esta característica para permitir carregar o programa simplesmente pressionando o botão "*upload*" no ambiente Arduino. Isto significa que o *bootloader* pode ter um *timeout* mais curto, já que a ativação do DTR (sinal baixo) pode ser bem coordenada com o início do *upload*.

Estas configurações têm outras implicações. Quando o Duemilanove está conectado a um computador rodando Mac OS X ou GNU/Linux, ele reinicia toda vez que a conexão é feita por software (via USB). No próximo meio segundo aproximadamente, o *bootloader* estará rodando no Duemilanove. Considerando que é programado para ignorar qualquer coisa a não ser um *upload* de um novo código, ele interceptará os primeiros bytes de dados sendo enviados para a placa depois que a conexão é aberta. Se um *sketch* rodando na placa recebe configurações de uma vez ou outros dados ao iniciar, assegure-se que o software que esteja comunicando espere um segundo depois de aberta a conexão antes de enviar estes dados.

O Duemilanove tem uma trilha que pode ser cortada para desabilitar o *auto-reset* e pode ser ressoldada para reativá-lo, é chamada de "RESET-EN", você pode também desabilitar o *auto-reset* conectando um resistor de 110 Ω dos +5 V até o sinal de reset.

2.8 Proteção contra sobrecorrente USB

O Arduino Duemilanove tem um polifusível que protege a porta USB do seu computador contra curtocircuito e sobrecorrente. Apesar da maioria dos computadores possuírem proteção interna própria, o fusível proporciona uma proteção extra. Se mais de 500 mA forem aplicados na porta USB, o fusível irá automaticamente interromper a conexão até que o curto ou a sobrecarga seja removida.

Capítulo 3

Referências da linguagem usada na programação do Arduino

Nesse capítulo iremos fazer uma pequena introdução sobre como são estruturados os programas para o Arduino, conhecendo a linguagem usada como referência e suas principais funções. E por fim veremos as funcionalidades extras que o uso de bibliotecas nos proporciona.

3.1 Linguaguem de referência

Os programas para o Arduino são implementados tendo como referência a linguagem C++. Preservando sua sintaxe clássica na declaração de variáveis, nos operadores, nos ponteiros, nos vetores, nas estruturas e em muitas outras características da linguagem. Com isso temos as referências da linguagem, essas podem ser divididas em três partes principais: As estruturas, os valores (variáveis e constantes) e as funções.

As estruturas de referências são:

- Estruturas de controle (if, else, break, ...)
- Sintaxe básica (define, include, ; , ...)
- Operadores aritméticos e de comparação (+, -, =, ==, !=, ...)
- Operadores booleanos (, ||, !)
- Acesso a ponteiros (*,)
- Operadores compostos (++, –, +=, ...)
- Operadores de bits (|, ’ , ...)

Os valores de referências são:

- Tipos de dados(byte, array, int , char , ...)
- Conversões(char(), byte(), int(), ...)
- Variável de escopo e de qualificação (variable scope, static, volatile, ...)
- Utilitários (sizeof(), diz o tamanho da variável em bytes)

É bom citar que o software que vem no Arduino já provê várias funções e constantes para facilitar a programação.

- setup()
- loop()
- Constantes (HIGH | LOW , INPUT | OUTPUT , ...)
- Bibliotecas (Serial, Servo, Tone, etc.)

3.2 Funções

As funções são referências essenciais para o desenvolvimento de um projeto usando o Arduino, principalmente para os iniciantes no assunto. Essas funções já implementadas e disponíveis em bibliotecas direcionam e exemplificam as funcionalidades básicas do microcontrolador. Temos como funções básicas e de referências as seguintes funções:

- Digital I/O

  pinMode() digitalWrite() digitalRead()
- Analógico I/O

  analogReference() analogRead() analogWrite() - PWM
- Avançado I/O

  tone() noTone() shiftOut() pulseIn()
- Tempo

  millis() micros() delay() delayMicroseconds()
- Matemática

  min() max() abs() constrain() map() pow() ***só do C/C++ sqrt() ***só do C/C++
- Trigonométrica

  sin() ***só do C/C++ cos() ***só do C/C++ tan() ***só do C/C++
- Números aleatórios

  randomSeed() random()
- Bits e Bytes

  lowByte() highByte() bitRead() bitWrite() bitSet() bitClear() bit()
- Interrupções externas

  attachInterrupt() detachInterrupt()
- Interrupções

  interrupts() noInterrupts()
- Comunicação Serial

3.3 Bibliotecas

O uso de bibliotecas nos proporciona um horizonte de programação mais amplo e diverso quando comparado a utilização apenas de estruturas, valores e funções. Isso é perceptível quando analisamos os assuntos que são abordados por cada biblioteca em específico. Lembrando sempre que, para se fazer uso de uma biblioteca esta já deve estar instalada e disponível na sua máquina. Temos as seguintes bibliotecas de referência:

- EEPROM - leitura e escrita de "armazenamento" permanente.
- Ethernet - para se conectar a uma rede Ethernet usando o Arduino Ethernet Shield.
- Firmata - para se comunicar com os aplicativos no computador usando o protocolo Firmata.
- LiquidCrystal - para controlar telas de cristal líquido (LCDs).
- Servo - para controlar servo motores.
- SPI - para se comunicar com dispositivos que utilizam bararmento Serial Peripheral Interface (SPI).
- SoftwareSerial - Para a comunicação serial em qualquer um dos pinos digitais.
- Stepper - para controlar motores de passo.
- Wire - Dois Wire Interface (TWI/I2C) para enviar e receber dados através de uma rede de dispositivos ou sensores.

Temos como referência também, o uso de bibliotecas mais específicas. O que é de extrema importância quando se faz o uso do arduino com um enfoque em uma determinada área. Como por exemplo:

Comunicação (redes e protocolos)

- Messenger - Para o processamento de mensagens de texto a partir do computador.
- NewSoftSerial - Uma versão melhorada da biblioteca `SoftwareSerial`.
- OneWire - Dispositivos de controle que usam o protocolo One Wire.
- PS2Keyboard - Ler caracteres de um PS2 teclado.
- Simple Message System - Enviar mensagens entre Arduino e o computador.
- SSerial2Mobile - Enviar mensagens de texto ou e-mails usando um telefone celular.
- Webduino - Biblioteca que cria um servidor Web (para uso com o Arduino Ethernet Shield).
- X10 - Envio de sinais X10 nas linhas de energia AC.
- XBee - Para se comunicar via protocolo XBee.
- SerialControl - Controle remoto através de uma conexão serial.

Sensoriamento

- Capacitive Sensing - Transformar dois ou mais pinos em sensores capacitivos.
- Debounce - Leitura de ruidos na entrada digital.

Geração de Freqüência e de Áudio

- Tone - Gerar ondas quadradas de freqüência de áudio em qualquer pino do microcontrolador.

Temporização

- DateTime - Uma biblioteca para se manter informado da data e hora atuais do software.
- Metro - Ajuda ao programador a acionar o tempo em intervalos regulares.
- MsTimer2 - Utiliza o temporizador de 2 de interrupção para desencadear uma ação a cada N milissegundos.

Utilidades

- TextString (String) - Manipular *strings*
- PString - uma classe leve para imprimir em buffers.
- Streaming - Um método para simplificar as declarações de impressão.

Capítulo 4

Instalação da IDE e suas bibliotecas

Neste capítulo iremos explicar como instalar a IDE e conectar a placa Arduino ao computador para sua programação. Junto com a placa arduino você deve ter um cabo USB tipo AB para poder conectá-lo ao computador.

4.1 Arduino para Windows

Primeiramente deve-se baixar o ambiente para o Arduino que pode ser encontrado no seguinte site: *//www.arduino.cc/en/Main/Software* , em download clique em Windows e baixe o arquivo arduino-0018.zip (ou mais novo), será necessário um programa capaz de descompactar o arquivo (exemplos: WinZip, WinRAR, etc.). Certifique-se de preservar a estrutura da pasta. Dê um duplo clique na pasta para abri-la, haverá uns arquivos e sub-pastas, clique no aplicativo arduino, este será seu ambiente de desenvolvimento.

Conecte a placa ao computador através do cabo USB, o LED verde na placa nomeado por PWR deve ascender, ele indica que a placa está ligada. O arduino seleciona automaticamente a fonte de alimentação adequada.

Quando se conecta a placa, o Windows deverá iniciar o processo de instalação do driver. No windows vista, o driver deve ser baixado e instalado automaticamente. No Windows XP o assistente Adicionar Novo Hardware será aberto:

- Quando o Windows perguntar se pode se conectar ao Windows Update para proucurar o software selecione não, clique em Avançar.

- Selecione personalizar, logo após selecione instalar e clique em Avançar.

- Certifique-se de poucurar o melhor driver, desmarque a pesquisa de mídia removível; selecione Incluir este local na pesquisa e proucure os drivers /FTDI USB Drivers diretórios de distribuição do Arduino. Clique em Avançar.

- O assistente irá proucurar o driver e em seguida, dizer que um hardware foi encontrado. Clique em Concluir.

- O assistente de novo hardware abrirá novamente, faça todos os passos da mesma maneira, desta vez, uma porta serial USB será encontrada.

4.2 Arduino para GNU/Linux

Para a instalação da IDE e suas bibliotecas no sistema operacional Linux, podemos assim como feito para o Windows baixar o arquivo zipado através do site *http://www.arduino.cc/en/Main/Software.* Apenas vale resaltar que neste sistema operacional temos um tratamento diferente com relação a manipulação de pastas e diretórios, agora o arquivo baixado é .tar.gz.

Além desta forma de instalação, temos uma outra mais objetiva para executar o mesmo procedimento, esta última usando apenas o terminal. Veremos um passo a passo deste procedimento usando o Linux Ubuntu.

Links usados:

- http://www.arduino.cc/playground/Linux/Ubuntu
- https://launchpad.net/ arduino-ubuntu-team/+archive/ppa
- https://launchpad.net/+help/soyuz/ppa-sources-list.html

Passo a Passo da instalação no Ubuntu

- O primeiro passo é com o terminal aberto digitar o comando: sudo add-apt-repository ppa:arduino-ubuntu-team/ppa
- Com o término do primeiro passo executamos o segundo comando digitando: sudo aptitude update
- E por fim executamos o último comando digitando: sudo aptitude install arduino
- Após estes três passos a IDE está instalada e pode ser acessada pelo menu aplicativos/desenvolvimento/arduino

Outras distribuições de Linux pesquisar no site *http://www.arduino.cc/en/Main/Software* e para Mac OS pesquisar em *http://www.arduino.cc/en/Guide/MacOSX*.

Capítulo 5

Desenvolvimento de Projetos

Neste capítulo iremos ver alguns exemplos de aplicações simples com o Arduino, agora com uma pequena base de c para arduino podemos começar a fazer e explicar exemplos mesmo para quem não possua uma grande infraestrutura possa realizá-lo.

5.1 Exemplo 1

Começaremos com o exemplo Blink, que já vem no aplicativo. Para encontrar o exemplo clique em File → Examples → Digital → Blink.

O programa tem como objetivo acender e apagar o LED de um em um segundo. Para compilar este exemplo não é necessário de nenhuma outra infraestrutura que não o próprio Arduino.
Primeiramente, vamos criar uma variável chamada `ledPin` que armazenará o número da porta onde o LED está conectado (variável do tipo inteiro):

```
int ledPin =  13;
```

Assim quando nos referirmos à variável `ledPin` estaremos nos referindo à saída 13.
O seguinte passo é classificar o `ledpin` como pino de Saída, isto é feito da seguinte maneira:

```
void setup() {
  pinMode(ledPin, OUTPUT);
}
```

A função `pinMode()` tem como primeiro parâmetro o pino e como segundo parâmetro se ele é pino de entrada ou saída. Agora começaremos a escrever o processamento. O programa rodará em um loop, pois não há ocorrências ou interferências que mudem o estado. Dentro do loop terá uma função que fará o LED ficar aceso por 1 segundo e depois ficar apagado por mais um segundo, após isso volta ao loop. Escreva da seguinte maneira:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

A função `digitalWrite()` escreve uma informação digital, ou seja, 0 (LOW) ou 1 (HIGH). Seu primeiro parâmetro é o pino de saída, que no caso é o `ledPin`. O segundo parâmetro é o estado, que no caso é a saída, HIGH ou LOW. Quando uma porta digital está em estado baixo (LOW), ela fica com 0V, já quando está em estado alto (HIGH), fica em 5 V.

A função `delay()` é um atraso que se dá para a continuação da leitura do programa, logo como foi escrito que o `ledPin` estará aceso, só após um segundo, ou como está escrito 1000 ms, irá ler a linha seguinte que escreve que a saída do `ledPin` é baixa, e o mesmo ocorre mais uma vez.

Antes de fazer o *upload* do programa, primeiro deve-se escolher a porta USB em que o Arduino se encontra. Para isso vá em Tools → Serial Port → *porta*, onde *porta* é o nome da porta onde está ligado o Arduino (/dev/ttyUSB*, no caso de GNU/Linux, COM* em Windows). Para saber em qual porta o Arduino se encontra, faça por tentativa e erro, logo escolha um e tente rodar, caso não rode, é o outro. Outro passo que deve-se fazer é escolher o modelo da placa, para isso vá em Tools → Board e o modelo da sua placa. Agora sim para fazer o *upload*, clique em Upload, como mostra a figura 5.1.

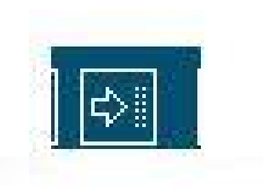


Figura 5.1: Upload

5.2 Exemplo 2

No segundo exemplo exploraremos melhor as saídas digitais. Neste exemplo serão necessários 10 LEDs para sua execução. Os LEDs ascenderão em sequência e ficarão acesos por 1 segundo, até que o último apagará e todos os outros em sequência apagarão.

Para a confecção do projeto ligue o positivo dos LEDs nos pinos digitais de 2 é 11 e a outra ponta em um *protoboard*, no lado negativo dos LEDs ligue resistores de 150 Ω, em série, e a outra ponta de todos os resistores no terra do Arduino, GND na placa. Assim como na figura 5.2.

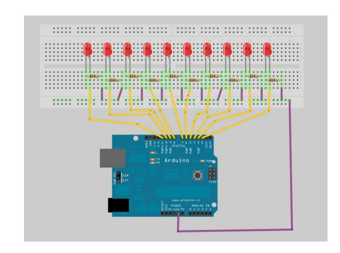


Figura 5.2: Circuito Exemplo 2

As primeiras linhas de comando são para declaração de variáveis, iremos precisar de uma variável constante e inteira de valor 10, em nosso caso chamamos de `ledCount`. Outra variável necessária é um vetor com 10 posições, enumerados de 2 é 11, que são os números dos pinos de saída digital que serão utilizados que também possuem valores inteiros, chamamos o vetor de `ledPins`. A seguir vem a declaração a ser feita:

```
const int ledCount = 10;

int ledPins[] = { 2, 3, 4, 5, 6, 7,8,9,10,11 };
```

Agora com as variáveis declaradas vamos definir o vetor `ledPins` como pinos de saída , para isso utilizaremos um *loop*, da seguinte maneira:

```
void setup() {
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    pinMode(ledPins[thisLed], OUTPUT);
  }
}
```

Na primeira posição do *for* declaramos uma variável que inicia em 0. Na segunda posição damos a condição para o *for*, e na terceira a cada vez que é verificada a condição do *for*, com execução da primeira, é acrescido 1 ao valor de `thisLed`, que é a variável que utilizaremos para chamar as determinadas posições do vetor `ledPins`. A função `pinMode()`, como vista no exemplo anterior, está declarando que o vetor `ledPins` é um vetor de saída de dados.

Agora será iniciado um *loop*, que fará com que o programa sempre rode, dentro dele terá um *for* que acenderá todos os LEDs sequencialmente, com um intervalo de 1 segundo (1000 ms) entre cada LED. O corpo do programa fica da seguinte maneira:

```
 void loop() {
```

```
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    digitalWrite(ledPins[thisLed], HIGH);
    delay(1000);
  }

  delay(1000);
```

Perceba que o *for* é da mesma maneira que o último, pois a idéia é sempre se referir às posições do vetor, a diferença aqui é que para cada posição estamos acendendo um LED, usando a função `digitalWrite`. A função `delay` foi utilizada para que seja mais fácil de visualizar que cada LED acende de cada vez, e que após todos os LEDs acenderem, todos ficam acesos por mais um segundo.

Agora será feito o mesmo, mas para apagar os LEDs, como mostra a seguir:

```
  for (int thisLed = 9; thisLed >= 0; thisLed--) {
     digitalWrite(ledPins[thisLed], LOW);
     delay(1000);
  }
  delay(1000);
}
```

A variável `thisLed`, utilizada apenas no *for*, começa com valor 9 e é decrescida de 1 até que chegue em 0, logo os LEDs apagarão do último a acender para o primeiro, e permanecerá apagado por 1 segundo.

Este foi o segundo exemplo, perceba que é possível modificá-lo com o que já foi aprendido, fazendo com que ele apague na mesma ordem que acende, ou fazendo qualquer outra mudança desejada.

5.3 Exemplo 3

Neste exemplo utilizaremos a entrada analógica e a saída serial na confecção de um capacímetro. Para isso será necessário um resistor, que pode ter qualquer valor que chamaremos de R1, um resitor de 220Ω, um capacitor, um *protoboard* e um fio.

Ligue o positivo do capacitor em um ponto comum e o negativo no terra, o resistor R1 entre o +5 da placa e o ponto comum, ligue o outro resistor, que chamaremos de R2 e tem o valor de 220 Ω entre o ponto comum e o pino 11, que é o pino que irá descarregar o capacitor. Ligue o fio do ponto comum a entrada analógica.

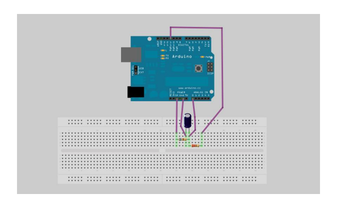


Figura 5.3: Circuito Exemplo 3

Iremos calcular o valor do capacitor medindo o tempo de carga, sabemos que uma constante de tempo ($\tau = TC$) em segundos é igual a resistência (R) em ohms multiplicado pela capacitância (C) em farads, e que a tensão no capacitor em uma constante de tempo (TC***) é de 63,2% do valor máximo, podemos calcular a capacitância, pois como sabemos o valor da fonte fornecida pelo arduino, 5 V, basta calcularmos 63,2% de sua tensão e quando a tensão no capacitor encontrar este valor basta dividi-la pelo valor da resistencia R1. $\tau = R * C$, ou seja $63,2\%V = R * C$, onde V é a tensão máxima.

A programação começa com a declaração de variáveis, ou seja, um pino analógico para medir tensão no capacitor, um pino de carga e um de descarga do capacitor e um pino com o valor de R1, como podemos ver no exemplo a seguir:

```
#define analogPin      0
#define chargePin      13
#define dischargePin   11
#define resistorValue  R1
unsigned long startTime;
unsigned long elapsedTime;
float microFarads;
float nanoFarads;
```

Perceba que o valor de R1 deve ser substituído pelo valor escolhido.

Devemos ajustar o pino 13 como pino de saída, como já foi feito em exemplos anteriores, e iniciar a saída serial a fim de depurar erros:

```
void setup(){
  pinMode(chargePin, OUTPUT);
  digitalWrite(chargePin, LOW);
  Serial.begin(9600);
}
```

No decorrer do desenvolvimento da apostila ainda não havíamos mencionado a comunicação serial, no próprio compilador arduino-0018 existe uma interface que lhe proporciona observar a saída e entrada na própria tela do computador, a figura 5.2 abaixo mostra onde ter acesso a esse recurso.

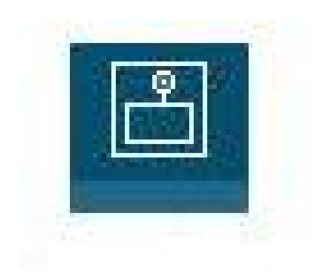


Figura 5.4: Comunicação Serial

Em nosso caso utilizamos a frequência de transferência de dados de 9600B/s, mas é possível selecionar outros valores já pré-determinados pelo programa que podem ser observados quando se abre a comunicação serial.

Iniciaremos o loop do programa fornecendo energia ao pino de carga do capacitor e acionando o *startTime*, que é a variável que irá temporizar o tempo de carga do capacitor. Para poder calcular os 63,2% da carga teremos que fazer uma conversão, pois o fim da escala é de 1023, logo 63,2% disso corresponde é 647, que é a porcentagem da tensão máxima no capacitor. Enquanto a entrada do pino analógico não equivaler a esta porcentagem de tempo não ocorre nada, apenas a contagem de tempo de carga do capacitor, que já esta sendo feita pela variável startTime. Quando esta porcentagem é ultrapassada mede-se a capacitância dividindo o tempo de carga pelo valor de R1.

Como é usual que valores de capacitância sejam baixos, na ordem de mili a nano Farad, é mais agradável que se expresse o valor em mili ou nano Farad, ou seja, multiplique o valor da capacitância por 1000 e acrescente o mF no final, caso mesmo com este procedimento o valor ainda seja menor que 1, podemos utilizar outras escalas (micro, nano, etc.). A programação referida pode ser observada a seguir:

```
void loop(){
  digitalWrite(chargePin, HIGH);  // coloque HIGH em chargePin
  startTime = millis();

  while (analogRead(analogPin) < 648) {
  }

  elapsedTime = millis() - startTime;
  microFarads = ((float)elapsedTime / resistorValue) * 1000;
  Serial.print(elapsedTime);
  Serial.println(" ms");

  if (microFarads > 1) {
    Serial.print((long)microFarads);
    Serial.println(" mF");
  }
  else {
    nanoFarads = microFarads * 1000.0;
    Serial.print((long)nanoFarads);
    Serial.println(" nF");
  }
```

O programa já está quase concluído, basta fazer a descarga do capacitor. Para isso “des-

ligue" o chargePin, ajuste dischargePin como saída, coloque LOW até que o capacitor esteja descarregado, e volte a ajustar o dischargePin como entrada, da seguinte maneira:

```
  digitalWrite(chargePin, LOW);
  pinMode(dischargePin, OUTPUT);
  digitalWrite(dischargePin, LOW);
  while (analogRead(analogPin) > 0) {
  }

  pinMode(dischargePin, INPUT);
}
```

5.4 Exemplo 4

Este exemplo ilustra o uso de uma das saídas PWM(Pulse-Width Modulation - Modulação por Largura de Pulso) do Arduino com um servomotor. Qualquer servo com um terminal de controle compatível pode ser utilizado. Aqui usaremos um polar rotor do tipo usado em antenas parabólicas. Primeiramente é importante saber o que é PWM e o que é possível fazer.
PWM é uma tecnologia que permite controlar o período cíclico da frequência da alimentação.

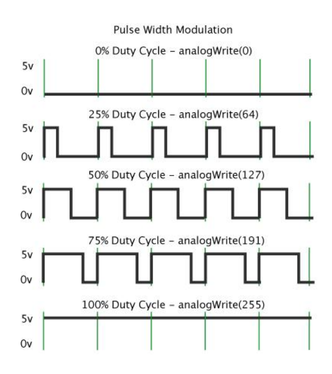


Figura 5.5: PWM

Suas aplicações são diversas e abrangem tanto usos industriais quanto domésticos. Em indústrias, o PWM pode ser usado para controlar elevadores de carga, esteiras rolantes e guinchos. Já em aplicações domésticas, pode ser usado para controle de iluminação, portões e cortinas. Iremos utilizar a entrada manual comandada por um potenciômetro linear de 100 kΩ.

O motor possui 3 fios: um vermelho, um preto e um branco. Os fios preto e vermelho correspondem ao negativo e positivo da alimentação, respectivamente, e neste exemplo podemos conectá-los diretamente aos pinos de alimentação do Arduino. O vermelho é conectado ao pino 5V, e o preto a qualquer um dos pinos GND. O fio branco é o terminal de controle, e deve ser conectado a uma das saídas digitais com PWM, qualquer um dos pinos 3, 5, 6, 9, 10 ou 11. No exemplo usaremos o 10.

O potenciômetro linear de 100 kΩ é conectado tendo um dos seus pinos extremos ligado ao GND, o outro extremo ao pino AREF, que fornece a tensão de referência, e o pino central conectado a qualquer uma das entradas analógicas, utilizaremos o pino 1.

Desta vez usaremos uma biblioteca para suporte de servos, logo no início do programa deveremos importá-la. Deveremos criar um objeto do tipo servo que será utilizado para controle, da seguinte maneira:

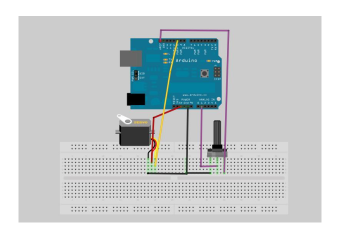


Figura 5.6: Exemplo 4

```
#include <Servo.h>
Servo meuservo;
```

A seguir é feita a parte de declaração de variáveis, iremos declarar um pino para o potenciômetro e servo, e uma variável inteira para o valor lido do potenciômetro. Também deveremos iniciar o servo, que em nosso caso está conectado ao pino 10, como vemos a seguir:

```
int potpin = 1;
int val;
void setup() {
  meuservo.attach(10);
}
```

Quando lermos a entrada do potenciômetro teremos que converter seu valor para graus para podermos controlar em graus quanto o motor irá girar, para isto utilizaremos a função map. Após isto apenas devemos enviar a informação para o motor e esperar alguns milissegundos para a movimentação do motor.

```
void loop() {
  val = analogRead(potpin);
  val = map(val, 0, 1023, 0, 180);
  meuservo.write(val);
  delay(500);
}
```

Capítulo 6

Referências Bibliográficas

Sites:

http://www.sabereletrônica.com.br/secoes/leitura/1307
Acessado em: 09/10/2010

http://arduino.cc/en/Reference/HomePage
Acessado em: 09/10/2010

http://www.arduino.cc
Acessado em: 09/10/2010

http://www.arduino.cc/en/Reference/AttachInterrupt
Acessado em: 09/10/2010

http://projeto39.wordpress.com/o-arduino-duemilanove/
Acessado em: 09/10/2010